



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

LOG3000 – INGÉNIERIE LOGICIELLE

TP4

Présenté à:

Mohammad Hamdaqa

Écrit par :

Cassy Charles (1947025)

Samy Checklat (1937812)

Remis le 17 Avril 2022

4.2. Questions

Q1. Dans chaque stratégie de déploiement ci-dessous, donnez la stratégie de déploiement appropriée qu'elle représente et un exemple concret.

Cas 1 : Stratégie de déploiement en continu

Dans ce cas, le déploiement en continu est utilisé. Un déploiement continu remplace lentement les instances de la version précédente d'une application par des instances de la nouvelle version de l'application. Pendant cette période, l'ancienne et la nouvelle version de l'application cohabitent ensemble sans toucher aux fonctionnalités ni affecter l'expérience utilisateur. Ce qu'on peut observer dans les figures données.

Exemple concret :

La mise à jour d'une application. Lorsque le déploiement d'une application se fait dans les conteneurs, la mise à jour se fait d'un conteneur à un autre. De cette manière, la dernière image du fournisseur d'applications est téléchargée et modifiée sur chaque conteneur. Si des problèmes de compatibilité se présentent, le conteneur peut être recréé avec l'ancienne image. De cette manière, les deux versions, anciennes et nouvelles coexistent ensemble jusqu'à la mise à jour complète.

Cas 2 :

i) **Stratégie Blue-Green:** Dans cette image, la couleur bleue représente l'environnement de production. C'est cet environnement qui reçoit le trafic des utilisateurs. Ce trafic est géré à l'aide d'un "load balancer" qui permet qu'une partie de l'environnement de production ne soit pas surchargé. La partie verte du dessin représente un clone de l'environnement de production. En revanche, ce clone n'est pas actif. De plus, ces deux environnements partagent les mêmes configurations ainsi que la même base de données.

ii) **A/B Testing** Dans cette image, la couleur verte représente la nouvelle version de l'application. Cet environnement permet de tester les performances ainsi que les fonctionnalités de la nouvelle version de l'application. Une fois les tests validés, le trafic de l'application de l'environnement bleu est dirigé vers l'environnement vert. L'environnement vert devient donc le nouvel environnement de production.

Exemple concret :

Concernant cette stratégie, il est possible de penser au routage du trafic. En effet, pour cette stratégie de déploiement, il faut mettre à jours les DNS CNAMEs pour les hôtes. Dans le cas contraire, il faut modifier les paramètres du “load balancer” afin d’apercevoir des changements immédiats. En exemple, certaines fonctionnalités, comme le drainage de connexions dans ELB

Cas 3 : Stratégie Canari

Pour cette stratégie, il est possible de remarquer que cette dernière ressemble à la stratégie “blue-green”. La différence vient du fait que la stratégie “Canari” utilise une seule étape tout en ayant une approche progressive.

Exemple concret :

Ce type de déploiement permet d’implémenter de nouvelles fonctionnalités de l’application dans une petite infrastructure de l’application. Lorsque cette fonctionnalité est approuvée, seul une petite partie des utilisateurs y accèdent. Cela permet de minimiser l’impact sur l’application.

Q2. Quelle est la relation entre OpenShift et Kubernetes?

Kubernetes et OpenShift présentent tous deux une architecture robuste et évolutive qui permet le développement, le déploiement et la gestion d'applications rapides et à grande échelle. Ils fonctionnent tous deux sous la licence Apache 2.0. Red Hat® OpenShift® est une distribution Kubernetes, un produit logiciel commercialisé dérivé d'un projet open source. Red Hat OpenShift et Kubernetes sont tous deux des logiciels d'orchestration de conteneurs, mais Red Hat OpenShift est conditionné en tant que plateforme open source d'entreprise en aval, ce qui signifie qu'il a subi des tests supplémentaires et qu'il contient des fonctionnalités supplémentaires qui ne sont pas disponibles dans le projet open source Kubernetes.

Q3. Dans vos propres mots, qu'est-ce que les Kubernetes et pourquoi est-il utile pour DevOps?

Kubernetes est une plateforme open source qui automatise les opérations de conteneurs Linux. Elle élimine une grande partie des processus manuels impliqués dans le déploiement et la mise à l'échelle des applications conteneurisées, ce qui veut dire qu'on peut regrouper des groupes d'hôtes exécutant des conteneurs Linux, et Kubernetes aide à gérer facilement et efficacement ces conteneurs.

L'une des principales raisons d'utiliser Kubernetes pour DevOps est qu'il réduit la charge de travail. Kubernetes améliore la qualité des processus DevOps avec la cohérence de l'environnement de développement, de test et de déploiement, la prise en charge de plusieurs frameworks, l'intégration d'une nouvelle application prend moins de temps, l'amélioration de la productivité et de l'efficacité des développeurs. Kubernetes facilite aussi la livraison rapide des logiciels avec une meilleure conformité, améliore la collaboration et la transparence au sein des équipes chargées de livrer le logiciel et réduit efficacement les coûts de développement et les risques de sécurité.

Q4. Expliquez la différence entre la mise à l'échelle horizontale et la mise à l'échelle verticale.

La mise à l'échelle horizontale fait référence à l'ajout de plusieurs périphériques ou nœuds informatiques au système pour améliorer les performances, tandis que la mise à l'échelle verticale correspond à l'ajout de ressources supplémentaires à un seul périphérique informatique afin d'améliorer les performances.

[1]

Q5. Qu'est-ce qu'un pod? Expliquez comment ils fonctionnent.

Les pods sont les plus petites unités déployables dans Kubernetes. Comme l'indique la documentation officielle : "Un pod est un groupe d'un ou plusieurs conteneurs, avec des ressources de stockage/réseau partagées, et une spécification sur la façon d'exécuter les conteneurs." Ainsi, dans les termes les plus simples possibles, un pod est le mécanisme qui permet d'activer un conteneur dans Kubernetes.

Les pods sont un produit des contrôleurs qui sont responsables de la gestion des opérations au sein du système Kubernetes. Les contrôleurs facilitent le déploiement, la réplication et l'état général des pods dans un cluster. Le contrôleur est le délégué de la réplication des pods si un pod échoue. Il existe trois principaux types de contrôleurs, à savoir les jobs, les déploiements et les statefulSets. Les travaux sont de courte durée, c'est-à-dire qu'ils n'existent que jusqu'à ce qu'un travail soit terminé. [2]

Q6. Qu'est-ce que "l'autoguérison d'application"?

Un système auto-réparateur peut découvrir des erreurs dans son fonctionnement et se modifier sans intervention humaine, ce qui lui permet de retrouver un meilleur état de fonctionnement. Dans les applications typiques, les problèmes sont documentés dans un "journal des exceptions" pour un examen ultérieur. La plupart des problèmes sont mineurs et peuvent être ignorés. Les problèmes graves peuvent nécessiter l'arrêt de l'application (par exemple, l'impossibilité de se connecter à une base de données qui a été mise hors ligne).

En revanche, les applications auto réparatrices intègrent des éléments de conception qui permettent de résoudre les problèmes. Par exemple, les applications qui utilisent Akka organisent les éléments dans une hiérarchie et attribuent les problèmes d'un acteur à son superviseur. Un grand nombre de bibliothèques et de frameworks de ce type facilitent les applications qui s'autoréparent par conception. [3]

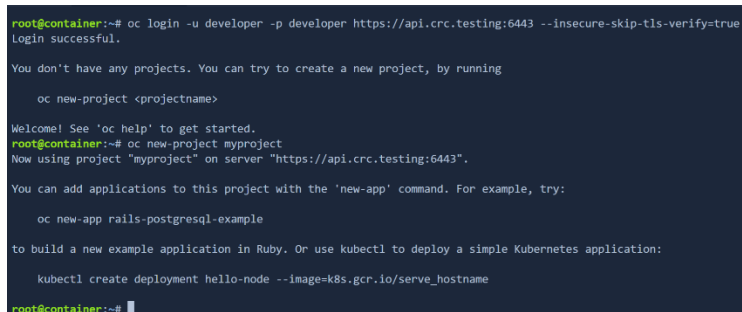
Q7. Quel est le but du routage?

La première et principale partie du routage consiste à déterminer le chemin par lequel les paquets vont passer de l'hôte émetteur, ou d'origine, à l'hôte récepteur, ou de destination. La deuxième partie du processus consiste à demander aux routeurs de faire passer les paquets d'un segment successif, ou "saut", du chemin au suivant jusqu'à ce que les paquets arrivent à leur destination. Le routage permet aussi de convertir un nom d'URL en une adresse IP, d'assurer le transfert sécurisé de fichiers sur Internet et de transférer le trafic sur la base des adresses MAC.

Q8. Dans l'interface du « portail d'apprentissage interactif », Sélectionnez deux scénarios: (1) Using the CLI to Manage Resource Objects (2) Transferring Files in and out of Containers Expliquez ce qui a été fait dans chaque scénario avec vos propres mots. Utilisez des captures d'écran pour appuyer votre réponse.

1. Using the CLI to Manage Resource Objects

Avec la commande: `oc login -u developer -p developer https://api.crc.testing:6443 --insecure-skip-tls-verify=true`, on se connecte en tant que développeur sur openshift. Et on crée un projet nommé myproject : `oc new-project myproject`.



```
root@container:~# oc login -u developer -p developer https://api.crc.testing:6443 --insecure-skip-tls-verify=true
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

Welcome! See 'oc help' to get started.
root@container:~# oc new-project myproject
Now using project "myproject" on server "https://api.crc.testing:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

    kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname

root@container:~#
```

Figure 1: Connexion et création d'un projet

Pour utiliser le cli pour gérer les objets ressources, nous devons d'abord déployer une application. Pour déployer notre application, nous exécutons la commande avec notre base de données PostgreSQL : `oc new-app postgresql-ephemeral -name database --param DATABASE_SERVICE_NAME=database --param POSTGRES_PASSWORD=password` . Nous surveillons alors le déploiement de l'application en exécutant : `oc rollout status dc/database`.

```
root@container:~# oc new-app postgresql-ephemeral --name database --param DATABASE_SERVICE_NAME=database --param POSTGRES_PASSWORD=password
warning: Cannot find git. Ensure that it is installed and in your path. Git is required to work with git repositories.
--> Deploying template "openshift/postgresql-ephemeral" to project myproject

PostgreSQL (Ephemeral)
-----
PostgreSQL database service, without persistent storage. For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/postgresql-container/.

WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing

The following service(s) have been created in your project: database.

Username: username
Password: password
Database Name: sampledb
Connection URL: postgresql://database:5432/

For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/postgresql-container/.

* With parameters:
  * Memory Limit=512Mi
  * Namespace=openshift
  * Database Service Name=database
  * PostgreSQL Connection Username=username
  * PostgreSQL Connection Password=password
  * PostgreSQL Database Name=sampledb
  * Version of PostgreSQL Image=10-e18

--> Creating resources ...
secret "database" created
service "database" created
deploymentconfig.apps.openshift.io "database" created
--> Success
Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
'oc expose service/database'
Run 'oc status' to view your app.
root@container:~# oc rollout status dc/database
Waiting for rollout to finish: 0 of 1 updated replicas are available...
Waiting for latest deployment config spec to be observed by the controller loop...
replication controller "database-1" successfully rolled out
root@container:~#
```

Figure 2: déploiement de l'application

Nous pouvons voir le nom des pods correspondant aux conteneurs en cours d'exécution pour cette application en exécutant : `oc get pods --selector name=database -o custom-columns`. La liste de tous les nœuds du cluster pour obtenir des informations contenant la base de données est alors affichée. Et avec la commande `oc rsh $POD`, on accède localement aux outils présents sur le système et il est possible de les gérer.

```

root@container:~# oc get pods --selector name=database
NAME          READY   STATUS    RESTARTS   AGE
database-1-fm5v2 1/1     Running   0           95s
root@container:~# POD=$(oc get pods --selector name=database -o custom-columns=NAME:.metadata.name --no-headers); echo $POD
database-1-fm5v2
root@container:~# oc rsh $POD
sh-4.4$ ps x
  PID TTY          STAT       TIME COMMAND
    1 ?        Ss          0:00 postgres
   58 ?        Ss          0:00 postgres: logger process
   60 ?        Ss          0:00 postgres: checkpointer process
   61 ?        Ss          0:00 postgres: writer process
   62 ?        Ss          0:00 postgres: wal writer process
   63 ?        Ss          0:00 postgres: autovacuum launcher process
   64 ?        Ss          0:00 postgres: stats collector process
   65 ?        Ss          0:00 postgres: bgworker: logical replication launcher
  150 pts/0    Ss          0:00 /bin/sh
  157 pts/0    R+          0:00 ps x
sh-4.4$ psql sampledb username
psql (10.17)
Type "help" for help.

sampledb=> \q
sh-4.4$ exit
exit
root@container:~#

```

2. Transferring Files in and out of Containers

Avec la commande: `oc login -u admin -p admin https://api.crc.testing:6443 --insecure-skip-tls-verify=true`, on se connecte en tant qu'admin sur openshift. Et on crée un projet nommé my project : `oc new-project myproject`.

```

[root@crc-dzk9v-master-0 /]# oc login -u admin -p admin https://api.crc.testing:6443 --insecure-skip-tls-verify=true
Login successful.

You have access to 64 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "default".
[root@crc-dzk9v-master-0 /]# oc new-project myproject
Now using project "myproject" on server "https://api.crc.testing:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

    kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname

[root@crc-dzk9v-master-0 /]#

```

Figure 3: Authentification et création d'un projet

Pour démontrer le transfert de fichiers depuis et vers un conteneur en cours d'exécution, nous devons d'abord déployer une application. Pour déployer notre application, nous exécutons la commande : `oc new-app openshiftkatacoda/blog-django-py --name blog`. Pour y accéder depuis un navigateur Web, nous devons également l'exposer en créant une route : `oc expose svc/blog`. Nous pouvons également surveiller le déploiement de l'application en exécutant : `oc rollout status dc/blog`. Cette commande se terminera

lorsque le déploiement sera terminé et que l'application web sera prête. Le résultat sera un conteneur en cours d'exécution. Nous pouvons voir le nom des pods correspondant aux conteneurs en cours d'exécution pour cette application en exécutant : `oc get pods --selector app=blog`.

```
[root@crc-dzk9v-master-0 /]# oc new-app openshiftkatakoda/blog-django-py --name blog
--> Found container image 927f823 (2 years old) from Docker Hub for "openshiftkatakoda/blog-django-py"

Python 3.5
-----
Python 3.5 available as container is a base platform for building and running various Python 3.5 applications and frameworks. Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Tags: builder, python, python35, python-35, rh-python35

* An image stream tag will be created as "blog:latest" that will track this image

--> Creating resources ...
imagestream.image.openshift.io "blog" created
deployment.apps "blog" created
service "blog" created
--> Success
Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
'oc expose service/blog'
Run 'oc status' to view your app.
[root@crc-dzk9v-master-0 /]# oc expose service/blog
route.route.openshift.io/blog exposed
[root@crc-dzk9v-master-0 /]# oc rollout status deployment/blog
deployment "blog" successfully rolled out
[root@crc-dzk9v-master-0 /]# oc get pods --selector deployment=blog
NAME                                READY    STATUS    RESTARTS   AGE
blog-5dc99d7545-qjfkp               1/1      Running   0           70s
```

Figure 4: Déploiement d'une application et nom des pods correspondants aux conteneurs

Pour les commandes suivantes qui doivent interagir avec ce pod, on utilise le nom du pod comme argument. Comme dans la photo ci-dessus, dans ce cas, le pod est `blog-5dc99d7545-qjfkp`. Pour créer un shell interactif dans le même conteneur que celui qui exécute l'application, on utilise la commande `oc rsh $POD`. À partir du shell interactif, on peut voir quels fichiers existent dans le répertoire de l'application avec la commande : `ls -las`.

```
[root@crc-dzk9v-master-0 /]# oc get pods --selector deployment=blog -o jsonpath='{.items[?(@.status.phase=="Running")].metadata.name}'
blog-5dc99d7545-qjfkp[root@crc-dzk9v-master-0 /]# pod() { local selector=$1; local query='?(@.status.phase=="Running")'; oc get pods --selector $selector - query}.metadata.name"; }
[root@crc-dzk9v-master-0 /]# POD='pod deployment=blog'; echo $POD
blog-5dc99d7545-qjfkp
[root@crc-dzk9v-master-0 /]# oc rsh $POD
(app-root)sh-4.2$ ls -las
total 80
0 drwxrwxr-x. 1 default root 52 Apr 17 01:35 .
0 drwxrwxr-x. 1 default root 28 Jun 18 2019 ..
4 -rwxrwxr-x. 1 default root 1454 Jun 18 2019 app.sh
0 drwxrwxr-x. 1 default root 43 Jun 18 2019 blog
0 drwxrwxr-x. 2 default root 25 Jun 18 2019 configs
4 -rw-rw-r--. 1 default root 230 Jun 18 2019 cronjobs.py
44 -rw-r--r--. 1 1000640000 root 44032 Apr 17 01:35 db.sqlite3
4 -rw-rw-r--. 1 default root 430 Jun 18 2019 Dockerfile
0 drwxrwxr-x. 2 default root 25 Jun 18 2019 htdocs
0 drwxrwxr-x. 1 default root 25 Jun 18 2019 katacoda
4 -rwxrwxr-x. 1 default root 806 Jun 18 2019 manage.py
0 drwxrwxr-x. 3 default root 20 Jun 18 2019 media
0 drwxrwxr-x. 1 default root 19 Apr 3 2019 .pki
4 -rw-rw-r--. 1 default root 832 Jun 18 2019 posts.json
8 -rw-rw-r--. 1 default root 7861 Jun 18 2019 README.md
4 -rw-rw-r--. 1 default root 203 Jun 18 2019 requirements.txt
4 -rw-rw-r--. 1 default root 1024 Apr 3 2019 .rnd
0 drwxrwxr-x. 4 default root 57 Jun 18 2019 .s2i
0 drwxrwxr-x. 4 default root 30 Jun 18 2019 static
0 drwxrwxr-x. 2 default root 148 Jun 18 2019 templates
```

Figure 5: Fichiers existant dans le répertoire de l'application

Pour l'application utilisée, cela a créé un fichier de base de données comme ceci : 44 -rw-rw-r-- 1 1000040000 root 44032 Apr 17 01:35 db.sqlite3. Pour confirmer dans quel répertoire se trouve le fichier, à l'intérieur du conteneur, on a fait la commande : pwd. Ce qui affiche :/opt/app-root/src. Pour quitter le shell interactif et revenir à la machine locale, on a fait exit. Maintenant, pour copier des fichiers du conteneur vers la machine locale, on a utilisé la commande oc rsync. Pour copier un seul fichier du conteneur vers la machine locale, on a fait : oc rsync \$POD:/opt/app-root/src/db.sqlite3 .

```
[root@crc-dzk9v-master-0 /]# cd /opt
[root@crc-dzk9v-master-0 opt]# mkdir db
mkdir: cannot create directory 'db': File exists
[root@crc-dzk9v-master-0 opt]# cd db
[root@crc-dzk9v-master-0 db]# oc rsync $POD:/opt/app-root/src/db.sqlite3 .
receiving incremental file list
db.sqlite3

sent 43 bytes received 44,130 bytes 88,346.00 bytes/sec
total size is 44,032 speedup is 1.00
[root@crc-dzk9v-master-0 db]# ls -las
total 44
0 drwxr-xr-x. 3 root root 37 Apr 17 01:49 .
0 drwxr-xr-x. 4 root root 70 Apr 17 01:46 ..
0 drwxr-x---. 3 root root 19 Apr 17 01:46 .kube
44 -rw-r--r--. 1 root root 44032 Apr 17 01:35 db.sqlite3
[root@crc-dzk9v-master-0 db]# oc rsync $POD:/opt/app-root/src/media .
receiving incremental file list
media/
media/images/

sent 32 bytes received 67 bytes 198.00 bytes/sec
total size is 0 speedup is 0.00
[root@crc-dzk9v-master-0 db]# mkdir uploads
[root@crc-dzk9v-master-0 db]# oc rsync $POD:/opt/app-root/src/media/. uploads
receiving incremental file list
./
images/

sent 31 bytes received 61 bytes 61.33 bytes/sec
total size is 0 speedup is 0.00
[root@crc-dzk9v-master-0 db]#
```

Figure 6: Copie des fichiers du conteneur vers la machine locale

On reçoit aussi la liste incrémentielle des fichiers : db.sqlite3 sent 43 bytes, received 44,130 bytes... On vérifie encore le contenu du répertoire actuel en exécutant la commande :ls -las. On voit alors que la machine locale a maintenant une copie du fichier : 44 -rw-rw-r-- 1 1000040000 root 44032 Apr 17 01:35 db.sqlite3. On crée notre répertoire uploads et on copie le répertoire media depuis le conteneur de notre pod sur la machine locale avec : oc rsync \$POD:/opt/app-root/src/media/. Uploads.

Maintenant pour le téléchargement de fichiers vers un conteneur :

Pour copier des fichiers de la machine locale vers le conteneur, on utilise encore la commande `oc rsync`. Contrairement à la copie du conteneur vers la machine locale, il n'y a pas de formulaire pour copier un seul fichier. Pour illustrer le processus de copie d'un seul fichier, on va inclure un fichier `robots.txt` dans un site web déployé. Tout d'abord, on crée un fichier `robots.txt` dans notre répertoire local qui contient : `User-agent : *`, `Disallow : /`. Pour l'application web utilisée, il héberge des fichiers statiques provenant du sous-répertoire `htdocs` du code source de l'application. On télécharge alors le fichier `robots.txt`.

```
[root@crc-diskv-master-0 ~]# export APP_ROUTE=oc get route blog --n myproject -o jsonpath="{http://{[.spec.host]}(/robots.txt)}"
[root@crc-diskv-master-0 ~]# curl $APP_ROUTE
ch1Not Found/1121p>The requested resource was not found on this server.</p>[root@crc-diskv-master-0 ~]# cd /opt/db
[root@crc-diskv-master-0 db]# pod { local.selector=di local.query="fig.status.phase='Running'"; oc get pods --selector $selector -o jsonpath="{.items[0].metadata.name}"; }
[root@crc-diskv-master-0 db]#
[root@crc-diskv-master-0 db]# POD=$(pod deployment-blob; echo $POD
blob-54c9p7d5-vf6p
[root@crc-diskv-master-0 db]# cat > robots.txt << !
> User-agent: *
> Disallow: /
> !
[root@crc-diskv-master-0 db]# oc rsync - $POD:/opt/app-root/src/htdocs --exclude= --include=robots.txt --no-perms
sanding incremental file list
robots.txt

sent 114 bytes received 35 bytes 112.67 bytes/sec
total size is 26 speedup is 0.15
[root@crc-diskv-master-0 db]# export APP_ROUTE=oc get route blog --n myproject -o jsonpath="{http://{[.spec.host]}(/robots.txt)}"
[root@crc-diskv-master-0 db]# curl $APP_ROUTE
User-agent: *
Disallow: /
[root@crc-diskv-master-0 db]# oc rsync - $POD:/opt/app-root/src/htdocs --no-perms
sanding incremental file list
db.sqlite3
.kube/
.kube/cache/
.kube/cache/discovery/
.kube/cache/discovery/api.crc.testing_6443/
.kube/cache/discovery/api.crc.testing_6443/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/admissionregistration.k8s.io/
.kube/cache/discovery/api.crc.testing_6443/admissionregistration.k8s.io/v1/
.kube/cache/discovery/api.crc.testing_6443/admissionregistration.k8s.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/apixextensions.k8s.io/
.kube/cache/discovery/api.crc.testing_6443/apixextensions.k8s.io/v1/
.kube/cache/discovery/api.crc.testing_6443/apixextensions.k8s.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/apiregistration.k8s.io/
.kube/cache/discovery/api.crc.testing_6443/apiregistration.k8s.io/v1/
.kube/cache/discovery/api.crc.testing_6443/apiregistration.k8s.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/apiserver.openshift.io/
.kube/cache/discovery/api.crc.testing_6443/apiserver.openshift.io/v1/
.kube/cache/discovery/api.crc.testing_6443/apiserver.openshift.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/apps.openshift.io/
.kube/cache/discovery/api.crc.testing_6443/apps.openshift.io/v1/
.kube/cache/discovery/api.crc.testing_6443/apps.openshift.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/apps/
.kube/cache/discovery/api.crc.testing_6443/apps/v1/
.kube/cache/discovery/api.crc.testing_6443/apps/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/authentication.k8s.io/
.kube/cache/discovery/api.crc.testing_6443/authentication.k8s.io/v1/
.kube/cache/discovery/api.crc.testing_6443/authentication.k8s.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/authorization.k8s.io/
.kube/cache/discovery/api.crc.testing_6443/authorization.k8s.io/v1/
.kube/cache/discovery/api.crc.testing_6443/authorization.k8s.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/authorization.openshift.io/
.kube/cache/discovery/api.crc.testing_6443/authorization.openshift.io/v1/
.kube/cache/discovery/api.crc.testing_6443/authorization.openshift.io/v1/serverresources.json
.kube/cache/discovery/api.crc.testing_6443/autoscaling.openshift.io/
.kube/cache/discovery/api.crc.testing_6443/autoscaling.openshift.io/v1/
```

Figure 7: Téléchargement du fichier web

```

[root@k8s-master-0 ~]# cd /opt/db
[root@k8s-master-0 ~]# pod() { local selector=$1; local query='?(@.status.phase="Running")'; oc get pods --selector $selector -o jsonpath="{.items[*].metadata.name}", }
[root@k8s-master-0 ~]# pod() pod deployment=blog; echo $POD
blog-5dc9d7545-qjfp
[root@k8s-master-0 ~]# git clone https://github.com/openshift-katacoda/blog-django-py
Cloning into 'blog-django-py'...
remote: Enumerating objects: 427, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 427 (delta 8), reused 0 (delta 0), pack-reused 409
Receiving objects: 100% (427/427), 75.28 KiB | 6.18 MiB/s, done.
Resolving deltas: 100% (208/208), done.
[root@k8s-master-0 ~]# oc rsync blog-django-py/. $POD:/opt/app-root/src --no-perms --watch &
[1] 139132
[root@k8s-master-0 ~]# sending incremental file list
.gitignore
Dockerfile
README.md
app.sh
cronjobs.py
manage.py
posts.json
requirements.txt
.git/
.git/HEAD
.git/config
.git/description
.git/index
.git/packed-refs
.git/branches/
.git/hooks/
.git/hooks/applypatch-msg.sample
.git/hooks/commit-msg.sample
.git/hooks/post-update.sample
.git/hooks/pre-applypatch.sample
.git/hooks/pre-commit.sample
.git/hooks/pre-merge-commit.sample
.git/hooks/pre-push.sample
.git/hooks/pre-receive.sample
.git/hooks/update.sample
.git/info/
.git/info/exclude
.git/logs/
.git/logs/HEAD
.git/logs/refs/
.git/logs/refs/heads/
.git/logs/refs/heads/master
.git/logs/refs/remotes/
.git/logs/refs/remotes/origin/
.git/logs/refs/remotes/origin/HEAD
.git/objects/
.git/objects/info/
.git/objects/pack/
.git/objects/pack/pack-df0855805da134742aaab0b478fa09fca733c.idx
.git/objects/pack/pack-df0855805da134742aaab0b478fa09fca733c.pack

```

Figure 8: Copie des fichiers

Lorsqu'on copie des fichiers dans le conteneur, il est nécessaire que le répertoire dans lequel les fichiers sont copiés existe et qu'il soit accessible en écriture à l'utilisateur ou au groupe qui exécute le conteneur. Les permissions sur les répertoires et les fichiers doivent être définies dans le cadre du processus de construction de l'image. Dans la commande ci-dessus, l'option `--no-perms` est utilisée, car le répertoire cible du conteneur, bien qu'accessible en écriture par le groupe sous lequel le conteneur est exécuté, appartient à un utilisateur différent. Cela signifie que, bien que les fichiers puissent être ajoutés au répertoire, les permissions sur les répertoires existants ne peuvent pas être modifiées. L'option `--no-perms` indique à `oc rsync` de ne pas essayer de mettre à jour les permissions ; cela évite qu'il échoue et renvoie des erreurs. Maintenant que le fichier `robots.txt` est téléchargé, la demande de ce fichier va être lancée. Cela a fonctionné sans qu'il soit nécessaire de prendre d'autres mesures, car le serveur utilisé pour héberger les fichiers statiques détecte automatiquement la présence d'un nouveau fichier dans le répertoire. Pour copier le contenu complet d'un répertoire dans le répertoire `htdocs` du conteneur, on exécute: `oc rsync images $POD:/opt/app-root/src/htdocs --no-perms`.

```
[root@crc-dzk9v-master-0 db]# jobs
[1]+  Running                  oc rsync blog-django-py/. $POD:/opt/app-root/src --no-perms --watch &
[root@crc-dzk9v-master-0 db]# export APP_ROUTE=$(oc get route blog -n myproject -o jsonpath='{http://'}[.spec.host]')
[root@crc-dzk9v-master-0 db]# curl $APP_ROUTE

<html>
  <head>
    <title>OpenShift Blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link href="//fonts.googleapis.com/css?family=Lalezar&subset=latin,latin-ext" rel="stylesheet" type="text/css">
    <link rel="stylesheet" href="/static/css/blog.css">
  </head>
  <body>
    <div class="page-header">
      <a href="/login/" class="top-menu"><span class="glyphicon glyphicon-user"></span></a>
      <h1><a href="/">OpenShift Blog</a></h1>
      <h2>blog-5d99d7545-qjfxp</h2>
    </div>
    <div class="content container">
      <div class="row">
        <div class="col-md-8">
          <div class="post">
            <div class="date">
              March 14, 2017, 3:26 a.m.
            </div>
            <h3><a href="/post/1/">What is OpenShift Origin</a></h3>
            <p>Origin is the upstream community project that powers OpenShift. Built around a core of Docker container packaging and Kubernetes container cluster management, Origin is also augmented by application lifecycle management functionality and DevOps</p>
          </div>

```

Figure 9: Vue du fichier Robots.txt

```
[root@crc-dzk9v-master-0 db]# echo "BLOG_BANNER_COLOR = 'blue'" >> blog-django-py/blog/context_processors.py
[root@crc-dzk9v-master-0 db]# sending incremental file list
blog/context_processors.py

sent 3,240 bytes  received 73 bytes  6,626.00 bytes/sec
total size is 227,043  speedup is 68.53
```

Figure 10: la liste incrémentielle des fichiers

```
[root@crc-dzk9v-master-0 db]# oc rsh $POD kill -HUP 1
[root@crc-dzk9v-master-0 db]# oc set env deployment/blog MOD_WSGI_RELOAD_ON_CHANGES=1
deployment.apps/blog updated
[root@crc-dzk9v-master-0 db]# oc rollout status deployment/blog
deployment "blog" successfully rolled out
[root@crc-dzk9v-master-0 db]# POD=$(oc get pods -l deployment=blog -o jsonpath='{.items[0].metadata.name}'); echo $POD
blog-85b4b444d7-8bp95
[root@crc-dzk9v-master-0 db]# jobs
[1]+  Running                  oc rsync blog-django-py/. $POD:/opt/app-root/src --no-perms --watch &
[root@crc-dzk9v-master-0 db]# kill -9 %1
[root@crc-dzk9v-master-0 db]# jobs
[1]+  Killed                  oc rsync blog-django-py/. $POD:/opt/app-root/src --no-perms --watch &
[root@crc-dzk9v-master-0 db]# oc rsync blog-django-py/. $POD:/opt/app-root/src --no-perms --watch &
[1] 133566
[root@crc-dzk9v-master-0 db]# sending incremental file list
.gitignore
Dockerfile
README.md
app.sh
cronjobs.py
manage.py
posts.json
requirements.txt
.git/
.git/HEAD
.git/config
.git/description
.git/index
```

Figure 11: Closure de tous les processus

```
[root@crc-dzk9v-master-0 persist]# oc rsh $POD ls -las /mnt
total 0
0 drwxrwx---. 3 root      root 19 Apr 17 04:11 .
0 dr-xr-xr-x. 1 root      root 61 Apr 17 04:10 ..
0 drwxr-x---. 3 1000640000 root 19 Apr 17 04:11 .kube
[root@crc-dzk9v-master-0 persist]# oc set volume deployment/dummy --remove --name=tmp-mount
deployment.apps/dummy volume updated
[root@crc-dzk9v-master-0 persist]# oc rollout status deployment/dummy
deployment "dummy" successfully rolled out
[root@crc-dzk9v-master-0 persist]# POD='pod deployment=dummy'; echo $POD
dummy-574985dc7f-4jg75
[root@crc-dzk9v-master-0 persist]# oc rsh $POD ls -las /mnt
total 0
0 drwxr-xr-x. 2 root root  6 Apr 11 2018 .
0 dr-xr-xr-x. 1 root root 61 Apr 17 04:12 ..
[root@crc-dzk9v-master-0 persist]# oc set volume deployment/dummy --add --name=tmp-mount --claim-name=data --mount-path /mnt
deployment.apps/dummy volume updated
[root@crc-dzk9v-master-0 persist]# oc rollout status deployment/dummy
deployment "dummy" successfully rolled out
[root@crc-dzk9v-master-0 persist]# POD='pod deployment=dummy'; echo $POD
dummy-5b644f7658-xfbxn
[root@crc-dzk9v-master-0 persist]# oc rsh $POD ls -las /mnt
total 0
0 drwxrwx---. 3 root      root 19 Apr 17 04:11 .
0 dr-xr-xr-x. 1 root      root 61 Apr 17 04:13 ..
0 drwxr-x---. 3 1000640000 root 19 Apr 17 04:11 .kube
[root@crc-dzk9v-master-0 persist]# oc delete all --selector deployment=dummy
pod "dummy-5b644f7658-xfbxn" deleted
replicaset.apps "dummy-574985dc7f" deleted
replicaset.apps "dummy-5b644f7658" deleted
replicaset.apps "dummy-67cfc5c656" deleted
[root@crc-dzk9v-master-0 persist]# oc get all --selector deployment=dummy -o name
pod/dummy-5b644f7658-lb28g
pod/dummy-5b644f7658-1l22q
replicaset.apps/dummy-5b644f7658
[root@crc-dzk9v-master-0 persist]# oc get pvc
NAME      STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
data      Bound   pv0026  100Gi     RWO,ROX,RWX               4m59s
[root@crc-dzk9v-master-0 persist]#
```

Figure 12: Efface des pods

Q9. Maintenant, vous devez implémenter le déploiement bleu-vert avec le code source disponible sur GitHub.

La commande “oc new-project..” permet de créer un nouveau projet sur OpenShift. Ce projet porte le nom de “bluegreen”. En revanche le nom affiché sera “Blue Green” avec une description “Blue Green Deployments”

La commande “oc new-app” permet de générer un nouvel objet OpenShift Entreprise. Cet objet permet de construire l’application, la déployer ainsi que la rouler. Le nom du “pod” pour ce projet sera “blue” tout en ayant une stratégie source. L’objet est généré à partir du répertoire GitHub de Cassyie.

```
[root@crc-dzk9v-master-0 /]# oc new-project bluegreen --display-name="Blue Green" --description='Blue Green Deployments'
Now using project "bluegreen" on server "https://api.crc.testing:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

    kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname

[root@crc-dzk9v-master-0 /]# oc new-app https://github.com/Ccassie/blue-green-openshift.git#master --name=blue --strategy=source
error: unable to checkout ref "master" in "https://github.com/Ccassie/blue-green-openshift.git" repository: fatal: couldn't find remote ref master
[root@crc-dzk9v-master-0 /]# oc new-app https://github.com/Ccassie/blue-green-openshift.git --name=blue --strategy=source
--> Found image b23116f (6 months old) in image stream "openshift/nodejs" under tag "14-ubi8" for "nodejs"

Node.js 14
-----
Node.js 14 available as container is a base platform for building and running various Node.js 14 applications and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Tags: builder, nodejs, nodejs14

* The source repository appears to match: nodejs
* A source build using source code from https://github.com/Ccassie/blue-green-openshift.git will be created
* The resulting image will be pushed to image stream tag "blue:latest"
* Use 'oc start-build' to trigger a new build

--> Creating resources ...
imagestream.image.openshift.io "blue" created
buildconfig.build.openshift.io "blue" created
deployment.apps "blue" created
service "blue" created
--> Success
Build scheduled, use 'oc logs -f buildconfig/blue' to track its progress.
Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
'oc expose service/blue'
Run 'oc status' to view your app.
[root@crc-dzk9v-master-0 /]#
```

Figure 13:Création du projet Blue Green sur OpenShift et du pod "blue"

Cette commande permet de rendre le "pod" par des personnes externes. Une route est créée et sera accessible aux personnes externes à partir de route.route.openshift.io/bluegreen

```
[root@crc-dzk9v-master-0 /]# oc expose service blue --name=bluegreen
route.route.openshift.io/bluegreen exposed
[root@crc-dzk9v-master-0 /]#
```

Figure 14: Création de la route accessible pour des personnes externes.

Afin de pouvoir apporter des modifications, nous avons cloner le répertoire du projet dans notre compte GitHub. Nous avons ensuite créé une branche portant le nom "green". Nous avons ensuite push cette nouvelle branche dans le répertoire

```
C:\Users\cassy\OneDrive\Desktop\Polytechnique Montreal\LOG3000\TP5\openshift\blue-green-openshift>git push --set-upstream origin green
info: please complete authentication in your browser...
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'green' on GitHub by visiting:
remote:   https://github.com/Ccassie/blue-green-openshift/pull/new/green
remote:
To https://github.com/Ccassie/blue-green-openshift.git
 * [new branch]      green -> green
Branch 'green' set up to track remote branch 'green' from 'origin'.

C:\Users\cassy\OneDrive\Desktop\Polytechnique Montreal\LOG3000\TP5\openshift\blue-green-openshift>
```

Figure 15: Ajout de la branche green dans le répertoire du projet

La commande “oc new-app” permet de générer un nouvel objet OpenShift Entreprise. Cet objet permet de construire l’application, la déployer ainsi que la router. Le nom du “pod” pour ce projet sera “green” tout en ayant une stratégie source. L’objet est généré à partir du répertoire GitHub de Cassie.

```
[root@crc-dzk9v-master-0 /]# oc new-app https://github.com/Ccassie/blue-green-openshift.git#green --name=green
--> Found image b23116f (6 months old) in image stream "openshift/nodejs" under tag "14-ubi8" for "nodejs"

Node.js 14
-----
Node.js 14 available as container is a base platform for building and running various Node.js 14 applications and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Tags: builder, nodejs, nodejs14

* The source repository appears to match: nodejs
* A source build using source code from https://github.com/Ccassie/blue-green-openshift.git#green will be created
* The resulting image will be pushed to image stream tag "green:latest"
* Use 'oc start-build' to trigger a new build

--> Creating resources ...
    imagestream.image.openshift.io "green" created
    buildconfig.build.openshift.io "green" created
    deployment.apps "green" created
    service "green" created
--> Success
Build scheduled, use 'oc logs -f buildconfig/green' to track its progress.
Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
'oc expose service/green'
Run 'oc status' to view your app.
[root@crc-dzk9v-master-0 /]#
```

Figure 16: Création du pod “green”

Cette commande permet de modifier la route initiale qui était associé au pod “blue” vers le pod “green”.

```
[root@crc-dzk9v-master-0 /]# oc get route/bluegreen -o yaml | sed -e 's/name: blue$/name: green/' | oc replace -f -
route.route.openshift.io/bluegreen replaced
[root@crc-dzk9v-master-0 /]#
```

Figure 17: Changement du service associé à la route du pod “blue” vers le pod “green”

Cette commande permet de modifier la route initiale qui était associé au pod “green” vers le pod “blue”.

```
root@crc-dzk9v-master-0 /]# oc get route/bluegreen -o yaml | sed -e 's/name: green$/name: blue/' | oc replace -f -
route.route.openshift.io/bluegreen replaced
root@crc-dzk9v-master-0 /]#
```

Figure 18: Changement du service associé à la route du pod "green" vers le pod "blue"

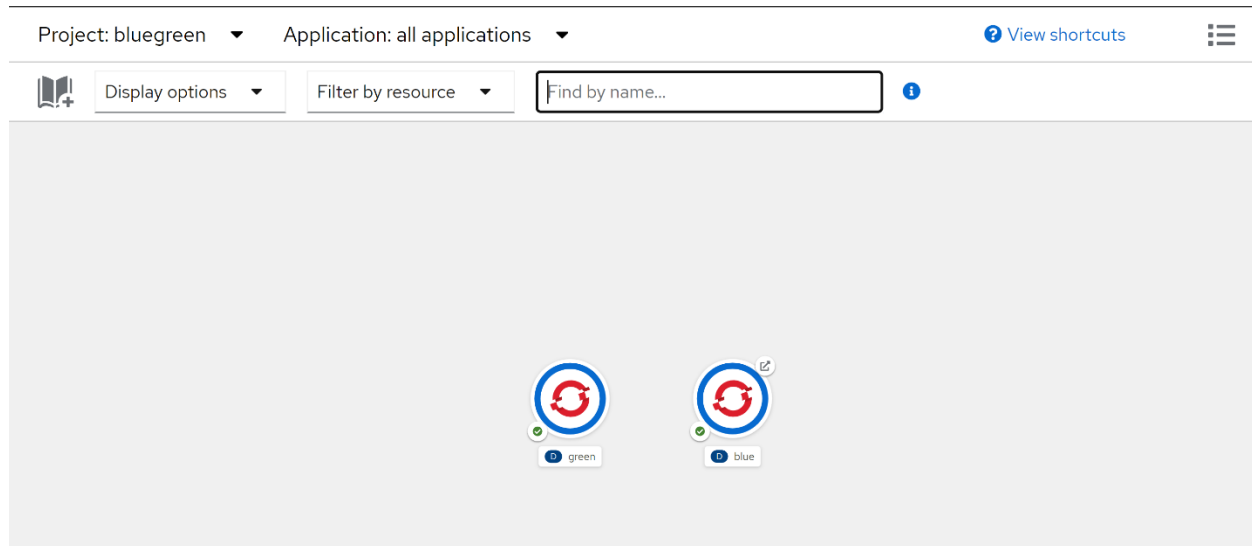


Figure 19: Affichage des pods fonctionnels

BuildConfigs

Create BuildConfig

Name	Labels	Created
BC blue	app=blue app.kubernetes.io/component=blue app.kubernetes.io/instance=blue	5 minutes ago
BC green	app=green app.kubernetes.io/component=green app.kubernetes.io/instance=green	3 minutes ago

Figure 20: Affichage des pods fonctionnels dans l'interface de configuration

4.3. Question de rétroaction

Nous travaillons à l'amélioration continue des travaux pratiques de LOG3000. Cette question peut être répondue très brièvement. Combien de temps avez-vous passé au travail pratique, en heures-

personnes, en sachant que deux personnes travaillant pendant trois heures correspondent à six heures-personnes ? Est-ce que l'effort demandé pour ce laboratoire est adéquat ?

Nous avons passé 8 heures-personnes sur ce travail pratique car la partie tutoriel nous a pris un peu de temps. L'effort demandé pour ce laboratoire est adéquat.

Références

- [1] «Quelle est la différence entre la mise à l'échelle horizontale et verticale,» Sawakinome, [En ligne].
Available: <https://fr.sawakinome.com/articles/technology/what-is-the-difference-between-horizontal-and-vertical-scaling.html>.
- [2] «How Kubernetes Pods Work,» 6 Février 2021. [En ligne]. Available:
<https://www.section.io/engineering-education/how-kubernetes-pods-work/>.
- [3] S. Ray, «What Are Self Healing Systems?,» 19 Février 2020. [En ligne]. Available:
<https://medium.com/lansaar/what-are-self-healing-systems-42ac9dd0e0aa>.